DZone REFCARDZ

BROUGHT TO YOU IN PARTNERSHIP WITH sysdig

Intro to **Docker Monitoring**

BY APURVA DAVÉ

OVERVIEW

Docker started as a tool for developers and test engineers to simplify software delivery, but it has rapidly evolved into a production-ready infrastructure platform. It promises to deliver software more flexibly and more scalably to your end users, while at the same time making microservices a reality.

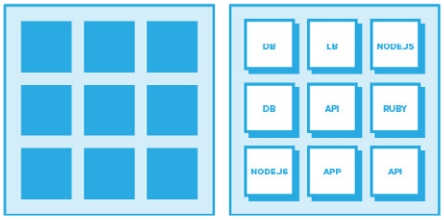
As any new platform moves into production, monitoring becomes an important aspect of its viability. That's especially true with a platform like Docker, where its architectural model actually changes how you need to instrument your systems in order to monitor it properly.

This Refcard will lay out the basics of the Docker monitoring challenge, give you hands on experience with basic monitoring options, and also spell out some more advanced options.

THE DOCKER MONITORING CHALLENGE

Containers have gained prominence as the building blocks of microservices. The speed, portability, and isolation of containers made it easy for developers to embrace a microservice model. There's been a lot written on the benefits of containers, so we won't recount it all here.

Containers are black boxes to most systems that live around them. That's incredibly useful for development, enabling a high level of portability from Dev through Prod, from developer laptop to cloud. But when it comes to operating, monitoring, and troubleshooting a service, black boxes make common activities harder, leading us to wonder: what's running in the container? How is the application code performing? Is it spitting out important custom metrics? From a DevOps perspective, you need deep visibility inside containers rather than just knowing that some containers exist.



great for development great for operations

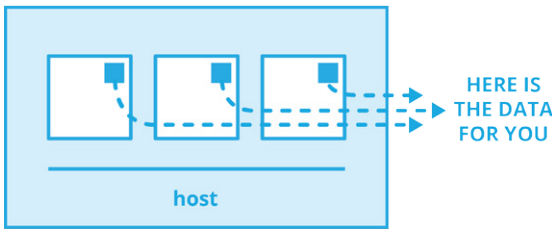
The typical process for instrumentation in a non-containerized environment—an agent that lives in the user space of a host or VM—doesn't work particularly well for containers. That's because containers benefit from being small, isolated processes with as few dependencies as possible. If you deploy the agent outside of the container, the agent can not easily see into the container to monitor the activity in the container. It will also require complex, brittle, and insecure networking among containers. If you deploy the agent inside the container, you have to modify each container to add the agent and deploy N agents for N containers. This increases dependencies and makes image management more difficult. And, at scale, running thousands of monitoring agents for even a modestly-sized deployment is an expensive use of resources.

ARCHITECTURAL MODELS FOR MONITORING CONTAINERS

Models for collecting instrumented data from containers do not stray too far afield from the past, and can generally be broken down into push and pull models. Push models have an agent that actively pushes metrics out to a central collection facility; pull models periodically query the monitoring target for the desired information.

As mentioned above, the most standard approach to infrastructure monitoring in a VM-based world is a push-based agent living in the user space. Two potential alternative solutions arise for containers: 1) ask your developers to instrument their code directly and push that data to a central collection point, or 2) leverage a transparent form of push-based instrumentation to see all application and container activity on your hosts.

Push-based
User Space Instrumentation



WHAT ARE YOUR DOCKER CONTAINERS DOING?

“Sysdig gives my team unprecedented visibility into our applications. We're using it extensively to help us solve complex issues in our most innovative products” – Adam Hertz, VP of Engineering at Comcast

We’re about to make your life easier.

Enterprises are moving to Docker containers for faster software development that promises developer agility, software portability, and scalable microservices. That’s all good news.

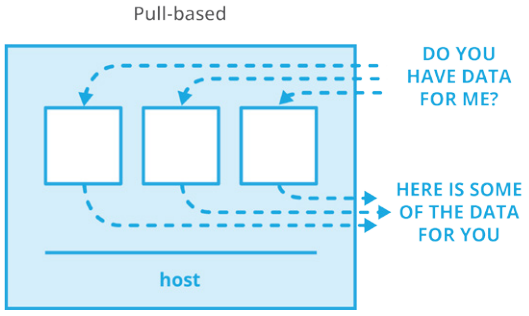
Here’s the bad news: Docker environments get harder to operate, because microservices and containers break legacy monitoring tools. Containers are great for developers, but they will wreak havoc on your DevOps team.

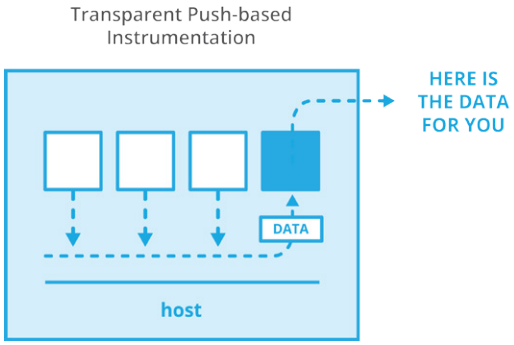
Worried? Relax, we got your back.

Sysdig is the first and only solution that can natively monitor Docker environments. Sysdig ContainerVision™ provides request-level visibility inside containers without invasive instrumentation. This approach overcomes the limitations of your old monitoring tools, and at the same time makes monitoring Docker simpler and more robust. All that means you can sleep at night when Kubernetes is auto-scaling your apps. Go on, press the snooze button.

Visit sysdig.com and find out!

sysdig





There is an additional, advanced topic that I'll touch on briefly in this Refcard: Docker containers often also use an orchestration to aggregate containers into services. These orchestration systems provide additional metadata that can be used to better monitor Docker. We will see an example later on of using Docker labels in this way to assist in service-level monitoring.

Let's now put some of this into practice with some common, open-source-based ways of gleaning metrics from Docker.

DOCKER MONITORING & TROUBLESHOOTING OPTIONS

There are of course a lot of commercial tools available that monitor Docker in various ways. For your purposes in getting started, it's more useful to focus on open-source Docker monitoring options. Not only will you be able to roll your sleeves up right away, you'll get a better understanding of the primitives that underpin Docker.

OPEN SOURCE TOOL	DESCRIPTION	PROS & CONS
Docker Stats API	Poll basic metrics directly from Docker Engine.	Basic stats output from CLI. No aggregation or visualization.
cAdvisor	Google-provided agent that graphs 1-minute data from the Docker Stats API.	Limited time-frame, limited metrics.
Time-series databases	Category of products like InfluxDB and Graphite that can store metrics data.	Good for historical trending. Requires you to set up a database, and glue together ingestion, DB, and visualization.
Sysdig	Container-focused Linux troubleshooting and monitoring tool.	Useful for deep troubleshooting and historical captures, but doesn't provide historical trending on its own.

DOCKER STATS API

Docker has one unified API, and in fact all commands you'd run from a CLI are simply tapping that endpoint.

For example, if you have a host running Docker, docker ps would return this, which is just a reformatting of API data.

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
8a9973a456b3        sysdig/sysdig-bot   "python bot.py"          20 hours ago
f561df9ef528        sysdig/agent        "/docker-entrypoint.s"   39 hours ago
90b3fa2a6bc2        wordpress           "/entrypoint.sh apach"   39 hours ago
11f18236aafe        mysql               "docker-entrypoint.sh"   39 hours ago
```

To show this let's query the API via curl and ask for all containers running. For brevity we're showing the JSON blob below for just one container, and prettied up the JSON.

```
curl --unix-socket /var/run/docker.sock http:/containers/json
  | python -m json.tool
```

```
{
        "Command": "python bot.py",
        "Created": 1470960101,
        "HostConfig": {
            "NetworkMode": "default"
        },
        "Id": "8a9973a456b3af0601c 44cf0ec19b35f01355622
  b5d5061552f5b84a6a335d25",
        "Image": "sysdig/sysdig-bot",
        "ImageID": "sha256:90d5bbf5afc4
  ce73223b5d57a249c5e05451f d4ab2414a799498e333503ffe4a",
        "Labels": {},
        "Mounts": [],
        "Names": [
            "/sysdig-bot"
        ],
        "NetworkSettings": {
            "Networks": {
                "bridge": {
                    "Aliases": null,
                    "EndpointID": "253f6015
  2b62e4d0a551657895bc84ec2e0e15657d f90d403f09ca6021425227",
                    "Gateway": "172.17.0.1",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "IPAMConfig": null,
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "Links": null,
                    "MacAddress": "02:42:ac:11:00:02",
                    "NetworkID": ""
                }
            }
        },
        "Ports": [],
        "State": "running",
        "Status": "Up 20 hours"
    },
```

Now let's apply this API to our monitoring needs. The /stats/ endpoint gives you streaming output of a wide selection of resource-oriented metrics for your containers. Let's get the available stats for just one container:

```
curl --unix-socket /var/run/docker.sock
  http:/containers/8a9973a456b3/stats

  ,"system\_cpu\_usage":266670930000000,"throttling\_
  data":},"cpu\_stats":,"system\_cpu\_
  usage":266671910000000,"throttling\_
  data":},"memory\_stats":{"usage":27516928,"max\_
  usage":31395840,"stats":{"active\_anon":17494016,"active\_
  file":5144576,"cache":10022912,
```

Not pretty, but an awful lot of metrics for us to work with!

If you wanted a one-shot set of metrics instead of streaming, use the stream=false option:

```
curl --unix-socket /var/run/docker.sock
http:/containers/8a9973a456b3/stats?stream=false
```

DOCKER MONITORING OVER TIME & IN-DEPTH

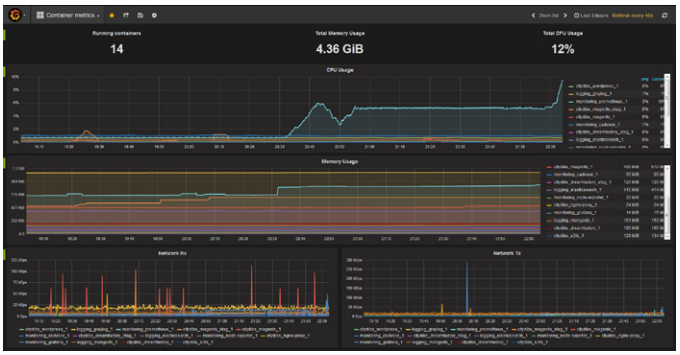
As you've probably guessed, the API is useful to get started but likely not the only thing you need to robustly monitor your applications running in Docker. The API is limiting in two ways: 1) it doesn't allow you to perform time-based trending and analysis, and 2) it doesn't give you the ability to do deep analysis on application- or system-level data. Let's attack these problems with cAdvisor and sysdig.

cAdvisor is a simple server that taps the Docker API and provides one minute of historical data in 1-second increments. It's a useful way to visualize what's going on at a high level with your Docker containers on a given host. cAdvisor simply requires one container per host that you'd like to visualize.

```
sudo docker run \
   --volume=/:/rootfs:ro \
   --volume=/var/run:/var/run:rw \
   --volume=/sys:/sys:ro \
   --volume=/var/lib/docker/:/var/lib/docker:ro \
   --publish=8080:8080 \
   --detach=true \
   --name=cadvisor \
   google/cadvisor:latest
```

cAdvisor is now running (in the background) on http://localhost:8080. The setup includes directories with Docker state cAdvisor needs to observe. Accessing the interface gives you this:

If you are looking to historically graph this data, you could also route data from cAdvisor to numerous time-series datastores via plugins, described here. Tying an open-source visualization engine on top of this, like Grafana, will allow you to produce something like this:



In most of these cases, however, we're limited to basic CPU, memory, and network data from these tools. What if we wanted to get deeper—to not only monitor resource usage, but processes, files, ports, and more?

DOCKER MONITORING AND DEEP TROUBLESHOOTING WITH SYSDIG

That's where another open-source tool, sysdig, comes into play. It's a Linux visibility tool with powerful command-line options that allow you to control what to look at and display it. You can also use csysdig, its curses-based interface, for an easier way to start. Sysdig also has the concept of chisels, which are pre-defined modules that simplify common actions.

Once you install sysdig as a process or a container on your machine, it sees every process, every network action, and every file action on the host. You can use sysdig "live" or view any amount of historical data via a system capture file.

As a next step, we can take a look at the total CPU usage of each running container:

```
\$ sudo sysdig -c topcontainers\_cpu
  CPU%            container.name
  --------------------------------------------------------
  90.13%          mysql
  15.93%          wordpress1
  7.27%           haproxy
  3.46%           wordpress2
  ...
```

This tells us which containers are consuming the machine's CPU. What if we want to observe the CPU usage of a single process, but don't know which container the process belongs to? Before answering this question, let me introduce the -pc (or -pcontainer) command-line switch. This switch tells sysdig that we are requesting container context in the output.

For instance, sysdig offers a chisel called topprocs_cpu, which we can use to see the top processes in terms of CPU usage. Invoking this chisel in conjunction with -pc will add information about

which container each process belongs to.

```
\$ sudo sysdig -pc -c topprocs\_cpu
```

```
CPU%            Process         Host_pid        Container_pid   container.name
--------------------------------------------------------------------------------
34.10%          mate-termi      3831            3831            host
20.88%          Xorg            2083            2083            host
14.99%          watch           11206           11206           host
13.04%          nessusd         1712            2334            host
9.02%           apache2         11277           44              wordpress4
9.02%           apache2         10113           46              wordpress1
8.98%           apache2         2382            45              wordpress2
8.78%           apache2         11040           44              wordpress3
8.61%           apache2         2381            44              wordpress2
5.90%           mysqld          1703            32              mysql
```

As you can see, this includes details such as both the external and the internal PID and the container name.

Keep in mind: -pc will add container context to many of the command lines that you use, including the vanilla sysdig output.

By the way, you can do all of these actions live or create a "capture" of historical data. Captures are specified by:

```
\$ sysdig -w myfile.scap
```

And then analysis works exactly the same.

What if we want to zoom into a single container and only see the processes running inside it? It's just a matter of using the same topprocs_cpu chisel, but this time with a filter:

```
\$ sudo sysdig -pc -c topprocs\_cpu container.name=client
  CPU%            Process         container.name
  ------------------------------------------------
  02.69%          bash            client
  31.04%          curl            client
  0.74%           sleep           client
```

Compared to docker top and friends, this filtering functionality gives us the flexibility to decide which containers we see. For example, this command line shows processes from all of the wordpress containers:

```
\$ sudo sysdig -pc -c topprocs\_cpu container.name contains
  wordpress
  CPU%            Process         container.name
  ------------------------------------------------
  6.38%           apache2         wordpress3
  7.37%           apache2         wordpress2
  5.89%           apache2         wordpress4
  6.96%           apache2         wordpress1
```

So to recap, we can:

- See every process running in each container including internal and external PIDs
- Dig down into individual containers
- Filter to any set of containers using simple, intuitive filters

…all without installing a single thing inside each container.

Now let's move on to the network, where things get even more interesting.

We can see network utilization broken up by process:

```
sudo sysdig -pc -c topprocs\_net
Bytes     Process    Host\_pid  Container\_pid   container.
name
--------------------------------------------------------
72.06KB   haproxy    7385       13            haproxy
56.96KB   docker.io  1775       7039          host
44.45KB   mysqld     6995       91            mysql
44.45KB   mysqld     6995       99            mysql
29.36KB   apache2    7893       124           wordpress1
29.36KB   apache2    26895      126           wordpress4
29.36KB   apache2    26622      131           wordpress2
29.36KB   apache2    27935      132           wordpress3
29.36KB   apache2    27306      125           wordpress4
22.23KB   mysqld     6995       90            mysql
```

Note how this includes the internal PID and the container name of the processes that are causing most network activity, which is useful if we need to attach to the container to fix stuff. We can also see the top connections on this machine:

```
sudo sysdig -pc -c topconns
Bytes        container.name Proto     Conn
-------------------------------------------------------------
22.23KB      wordpress3     tcp        172.17.0.5:46955->
172.17.0.2:3306
22.23KB      wordpress1     tcp        172.17.0.3:47244->
172.17.0.2:3306
22.23KB      mysql          tcp        172.17.0.5:46971->
172.17.0.2:3306
22.23KB      mysql          tcp        172.17.0.3:47244->
172.17.0.2:3306
22.23KB      wordpress2     tcp        172.17.0.4:55780->
172.17.0.2:3306
22.23KB      mysql          tcp        172.17.0.4:55780->
172.17.0.2:3306
14.21KB      host           tcp        127.0.0.1:60149->
127.0.0.1:80
```

This command line shows the top files in terms of file I/O, and tells you which container they belong to:

```
\$ sudo sysdig -pc -c topfiles\_bytes
Bytes   container.name Filename
-------------------------------------------------------------
63.21KB  mysql     /tmp/\#sql\_1\_0.MYI
6.50KB   client    /lib/x86\_64-linux-gnu/libc.so.6
3.25KB   client    /lib/x86\_64-linux-gnu/libpthread.so.0
3.25KB   client    /lib/x86\_64-linux-gnu/libgcrypt.so.11
3.25KB   client    /usr/lib/x86\_64-linux-gnu/libwind.so.0
3.25KB   client    /usr/lib/x86\_64-linux-gnu/libgssapi\_
krb5.so.2
3.25KB   client    /usr/lib/x86\_64-linux-gnu/liblber-
2.4.so.2
3.25KB   client    /lib/x86\_64-linux-gnu/libssl.so.1.0.0
3.25KB   client    /usr/lib/x86\_64-linux-gnu/libheimbase.
so.1
3.25KB   client    /lib/x86\_64-linux-gnu/libcrypt.so.1
```

Naturally there is a lot more you can do with a tool like this, but that should be a sufficient start to put our knowledge to work in some real-life examples.

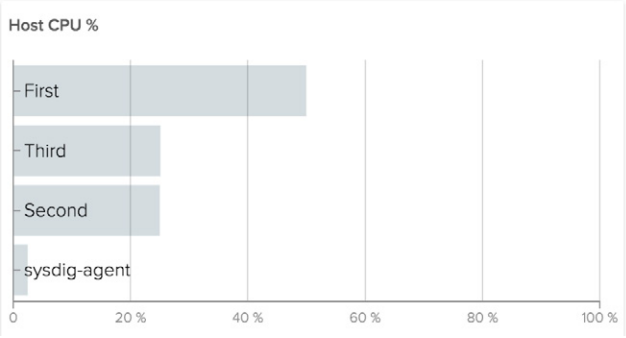
REAL-WORLD EXAMPLES: WHAT TO MONITOR, WHY, AND HOW

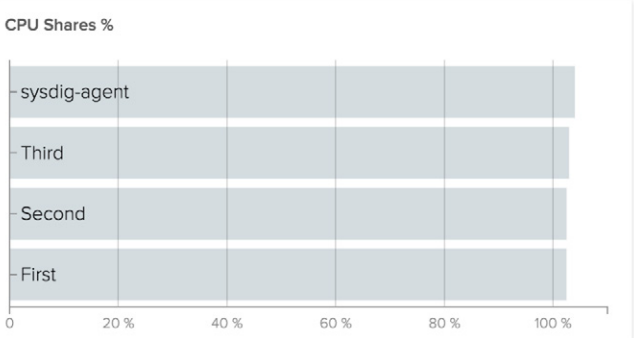
So now we've done some of the basics, and it's time to take the training wheels off. Let's take a look at some more complex, real-world metrics you should pay attention to. We'll show you the metrics, talk about why they're important and what they might mean. For this section we've visualized the data using Sysdig Cloud, the commercial version of Sysdig that's designed to aggregate data across many hosts and display within a web UI. You could do the following examples via any of the open-source time-series databases, provided you're collecting the correct information.

VISUALIZING CPU SHARES & QUOTA

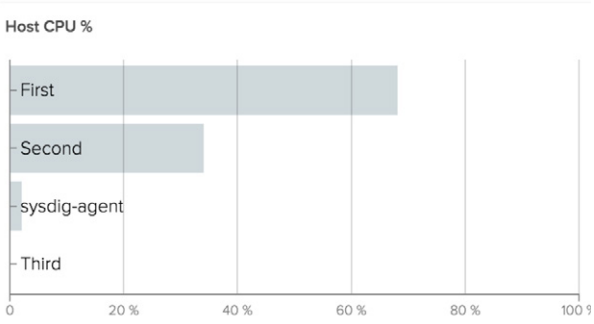
For those of you used to monitoring in a VM-based world, you're likely familiar with the concepts of CPU allocation, stolen CPU, and greedy VMs. Those same issues apply with containers, except they are magnified significantly. Because you may be packing containers densely on a machine, and because workloads are typically much more dynamic than in VM-based environments, you may encounter significantly more resource conflict if you're not carefully monitoring and managing allocation. Let's focus on CPU, as it's a bit more complex than memory.

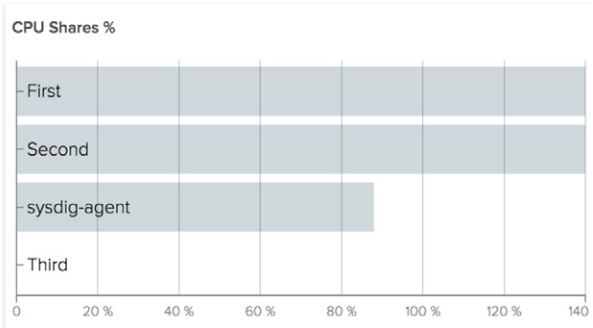
Let's start by visualizing CPU shares. Imagine a host with 1 core and 3 containers using as much CPU as possible. We assign 1024 shares to one container and 512 shares to the other two. This is what we get:





First is using 2 times the Host CPU than the others because it has 2 times the shares. All of them are using 100% of CPU shares assigned. But what happens if *Third* does not need any CPU at all?





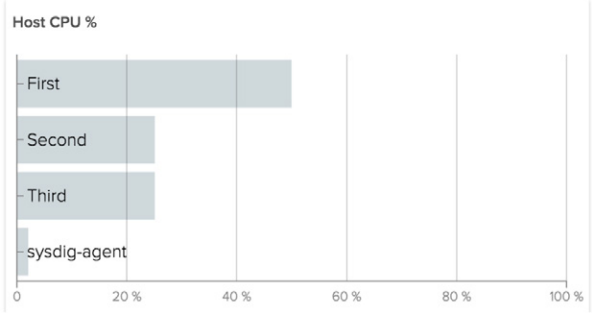
The amount of unused shares is given to others relative to their weight. So if *Third* is not using any of its CPU shares, *First* and *Second* instead are using 140% of CPU Shares. In general, it's OK to consume more shares than originally allocated, because the kernel tries not to waste CPU.

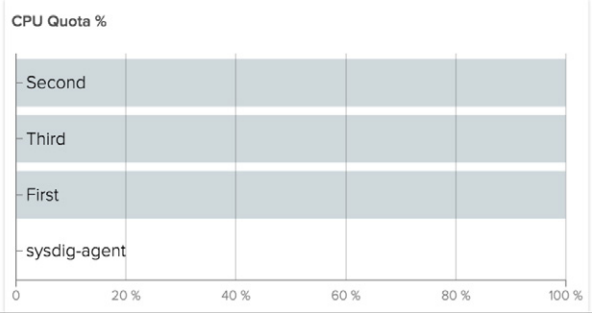
A percentage of shares used that's consistently over 100 means we are not allocating enough resources to our services. The implication in the example above is that *First* and *Second* were able to consume much more CPU than they were originally allocated. If either of those were, for example, a web server, it likely means we are allocating less CPU than it needs to complete current user requests (that's not a good situation). If either were a batch processing job, it means the job can use more CPU to finish faster (good, but maybe not critical).

VISUALIZING CPU QUOTA

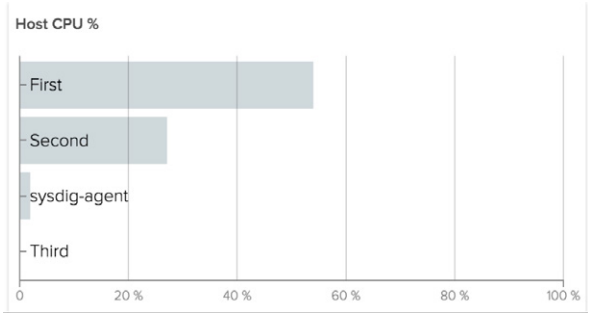
Giving processes the maximum available CPU may be not always be what you want. If your cluster is multi-tenant, or if you just need a safe ceiling for an unpredictable application, you might like to implement a hard limit on CPU utilization. The Linux kernel supports absolute CPU limits with CPU quotas. You assign a quota in milliseconds relative to a period, and the process will be able to spend on CPU only that fraction of time in a period.

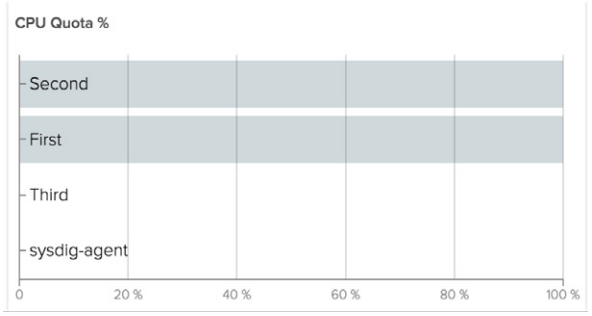
For example let's consider the same case as above, now with a quota of 50ms/100ms for *First* and 25ms/100ms for *Second* and *Third*:





The result is the same as with shares. The difference occurs when *Third* does not use the CPU allocated to it.



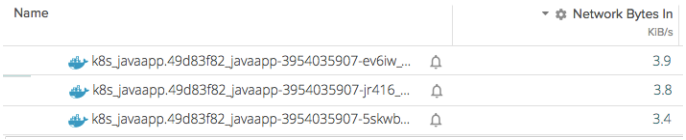


Now instead of giving CPU to other containers, the kernel is enforcing the absolute quota given. The total CPU usage we will see reported for the host will be 75%.

BASIC NETWORKING DATA

Regardless of your platform, some things don't change… and that's certainly true when it comes to networking data. Especially with Docker in the mix, networking can become more complex and communication patterns can become more convoluted. It's important to keep track of basic information, such as how much data is a container consuming? Emitting?

This type of data collection requires something more full-featured than the Docker API, so instead you could collect this type of information from open-source sysdig. Let's look at some basic network data for a set of three containers each running the same Java application:



As you can see. there is some slight variation among these three containers. If, however, we saw an extreme variation, we may want to investigate further.

At the same time, since these containers are all running the same Java application, it may be more useful to consider them a "service" and see how they are performing in aggregate. This leads up to our last example.

FROM CONTAINER TO MICROSERVICE DATA WITH LABELS

Docker provides a concept called "labels." These are much like they sound—additional, contextual information is are applied on a per-container basis. They are unstructured and non-hierarchical. As such, you can use them to broadly identify subcategories of your containers. All the containers of a given service could carry the same label, non-standard containers could carry another label, different versions of software could have yet another label. If you're a filer and an organizer, labels will be heaven for you.

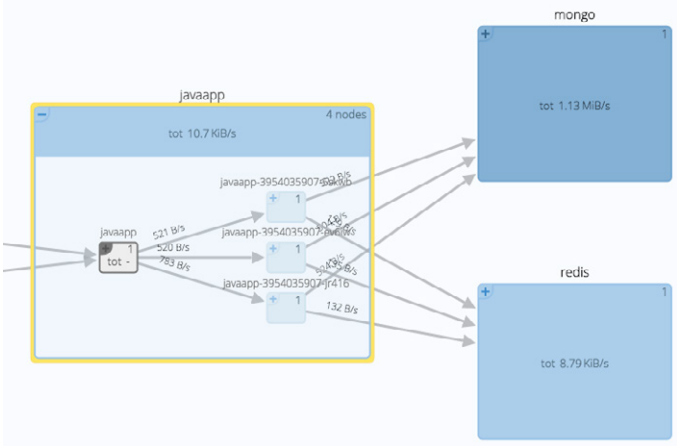
So what can we do with a label? Well, the first thing is that we can aggregate data. From the example above, let's suppose we applied the label "javapp" to those three containers. Now, when we show our network data we see something much simpler:

One line—that's it. In this case we're showing the average network data across all three containers, but you could easily calculate anything that helps you better understand the performance of this collection of containers.

But let's go a little further with labels, network data, and the "top connections" example we showed in the open-source section.

Using this information and an appropriate visualization, we can do more than create a table of network data: we can actually create a map of our services, the containers that make them up, and who they are communicating with. Here we can see the aggregated java service, the individual containers that make up the service, and (in a more complete view) would show all the other services in your environment that the java service communicates with. Note that this is a little more advanced than the other examples, and

in particular the visualization may require some coding in D3 or something similar if you want to stay fully open source.



Here we see a few different things: our "javaapp" consists of three containers (blue) and a service called "javapp" (grey), which is just an abstraction created by whoever is routing requests to those containers. We see each of those containers communicating with a Mongo service and a Redis service, and presumably those are made up of containers as well (hidden here to avoid too much complexity).

This view helps us in a few different ways:

- We quickly can understand the logical composition of our application.
- We can aggregate containers into higher-level services.
- We can easily see communication patterns among containers.
- We may be able to easily spot outliers or anomalies.

CONCLUSION

In this Refcard, we've walked from first principles using the Docker Stats API all the way up to more complex analysis of our system's performance. We've used data sources such as cAdvisor and sysdig to analyze real-world use cases such as greedy containers or mapping network communication.

As you can see, Docker monitoring can start very simply but grow complex as you actually take containers into production. Get experience early and then grow your monitoring sophistication to what your environment requires.

ABOUT THE AUTHOR

APURVA DAVÉ @ApurvaBDave is the VP of marketing at Sysdig. He's in marketing and (gasp!) not afraid of a command line. He's been helping people analyze and accelerate infrastructure for the better part of two decades. He previously worked at Riverbed on both WAN acceleration and Network Analysis products, and at Inktomi on infrastructure products. He has a computer science degree from Brown University and an MBA from UC Berkley.

RESOURCES

Docker Stats Documentation:
https://docs.docker.com/engine/reference/api/docker_remote_api

Sysdig Open Source Documentation:
http://www.sysdig.org/wiki

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more.

"DZone is a developer's dream," says PC Magazine.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513
888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795

BROUGHT TO YOU IN PARTNERSHIP WITH